

# Boosting with Averaged Weight Vectors

Nikunj C. Oza

Computational Sciences Division  
NASA Ames Research Center  
Mail Stop 269-3  
Moffett Field, CA 94035-1000, USA  
oza@email.arc.nasa.gov

**Abstract.** AdaBoost [5] is a well-known ensemble learning algorithm that constructs its constituent or *base* models in sequence. A key step in AdaBoost is constructing a distribution over the training examples to create each base model. This distribution, represented as a vector, is constructed to be orthogonal to the vector of mistakes made by the previous base model in the sequence [7]. The idea is to make the next base model's errors uncorrelated with those of the previous model. Some researchers have pointed out the intuition that it is probably better to construct a distribution that is orthogonal to the mistake vectors of *all* the previous base models, but that this is not always possible [7]. We present an algorithm that attempts to come as close as possible to this goal in an efficient manner. We present experimental results demonstrating significant improvement over AdaBoost and the Totally Corrective boosting algorithm [7], which also attempts to satisfy this goal.

## 1 Introduction

AdaBoost [5] is one of the most well-known and highest-performing ensemble classifier learning algorithms [4]. It constructs a sequence of base models, where each model is constructed based on the performance of the previous model on the training set. In particular, AdaBoost calls the base model learning algorithm with a training set weighted by a distribution.<sup>1</sup> After the base model is created, it is tested on the training set to see how well it learned. We assume that the base model learning algorithm is a *weak learning algorithm* [6]; that is, with high probability, it produces a model whose probability of misclassifying an example is less than 0.5 when that example is drawn from the same distribution used to generate the training set. The point is that such a model performs better than random guessing.<sup>2</sup> The weights of the correctly classified examples and

<sup>1</sup> If the base model learning algorithm cannot take a weighted training set as input, then one can create a sample with replacement from the original training set according to the distribution and call the algorithm with that sample.

<sup>2</sup> The version of AdaBoost that we use was designed for two-class classification problems. However, it is routinely used for a larger number of classes when the base model learning algorithm is strong enough to have an error less than 0.5 in spite of the larger number of classes.

misclassified examples are scaled down and up, respectively, so that the two groups' total weights are 0.5 each. The next base model is generated by calling the learning algorithm with this new weight distribution and the training set. The idea is that, because of the weak learning assumption, at least some of the previously misclassified examples will be correctly classified by the new base model. Previously misclassified examples are more likely to be classified correctly because of their higher weights, which focus more attention on them. Kivinen and Warmuth [7] have shown that AdaBoost scales the distribution with the goal of making the next base model's mistakes uncorrelated with those of the previous base model. It is well-known that ensembles need to have low correlation in their base models' errors in order to perform well [11].

Given this point, we would think, as was pointed out in [7], that AdaBoost would perform better if the next base model's mistakes were uncorrelated with those of *all* the previous base models instead of just the previous one. It is not always possible to construct a distribution consistent with this requirement. However, we can attempt to find a distribution that comes as close as possible to satisfying this requirement. Kivinen and Warmuth [7] devised the *Totally Corrective* boosting algorithm, which attempts to do this. However, they do not present any empirical results. Also, they hypothesize that this algorithm will overfit and; therefore, not perform well. This paper presents a new algorithm, called AveBoost, which has the same goal as the Totally Corrective algorithm. In particular, AveBoost calculates the next base model's distribution by first calculating a distribution the same way as in AdaBoost, but then averaging it elementwise with those calculated for the previous base models. In this way, AveBoost attempts to take all the previous base models into account in constructing the next model's distribution. In Section 2, we review AdaBoost and describe the Totally Corrective algorithm. In Section 3, we state the AveBoost algorithm and describe the sense in which our solution is the best one possible. In Section 4, we present an experimental comparison of AveBoost with AdaBoost and the Totally Corrective algorithm. Section 5 summarizes this paper and describes ongoing and future work.

## 2 AdaBoost and Totally Corrective Algorithm

Figure 1 shows AdaBoost's pseudocode. AdaBoost constructs a sequence of base models  $h_t$  for  $t \in \{1, 2, \dots, T\}$ , where each one is constructed based on the performance of the previous base model on the training set. In particular, AdaBoost maintains a distribution over the  $m$  training examples. The distribution  $\mathbf{d}_1$  used in creating the first base model gives equal weight to each example ( $d_{1,i} = 1/m$  for all  $i \in \{1, 2, \dots, m\}$ ). AdaBoost now enter the loop, where the base model learning algorithm  $L_b$  is called with the training set and  $\mathbf{d}_1$ .<sup>3</sup> The returned model  $h_1$  is then tested on the training set to see how well it learned. Training

<sup>3</sup> As mentioned earlier, if  $L_b$  cannot take a weighted training set as input, then we can give it a sample drawn with replacement from the original training set according to the distribution  $\mathbf{d}$  induced by the weights.

**AdaBoost**( $\{(x_1, y_1), \dots, (x_m, y_m)\}, L_b, T$ )  
Initialize  $d_{1,i} = 1/m$  for all  $i \in \{1, 2, \dots, m\}$ .  
For  $t = 1, 2, \dots, T$ :  
 $h_t = L_b(\{(x_1, y_1), \dots, (x_m, y_m)\}, \mathbf{d}_t)$ .  
Calculate the error of  $h_t : \epsilon_t = \sum_{i: h_t(x_i) \neq y_i} d_{t,i}$ .  
If  $\epsilon_t \geq 1/2$  then,  
set  $T = t - 1$  and abort this loop.  
Calculate distribution  $\mathbf{d}_{t+1}$ :

$$d_{t+1,i} = d_{t,i} \times \begin{cases} \frac{1}{2(1-\epsilon_t)} & \text{if } h_t(x_i) = y_i \\ \frac{1}{2\epsilon_t} & \text{otherwise.} \end{cases}$$

**Output** the final hypothesis:  
 $h_{fin}(x) = \operatorname{argmax}_{y \in Y} \sum_{t: h_t(x)=y} \log \frac{1-\epsilon_t}{\epsilon_t}$ .

**Fig. 1.** AdaBoost algorithm:  $\{(x_1, y_1), \dots, (x_m, y_m)\}$  is the training set,  $L_b$  is the base model learning algorithm, and  $T$  is the maximum allowed number of base models.

**Totally Corrective AdaBoost**( $\{(x_1, y_1), \dots, (x_m, y_m)\}, L_b, T$ )  
Initialize  $d_{1,i} = 1/m$  for all  $i \in \{1, 2, \dots, m\}$ .  
For  $t = 1, 2, \dots, T$ :  
 $h_t = L_b(\{(x_1, y_1), \dots, (x_m, y_m)\}, \mathbf{d}_t)$ .  
Calculate the mistake vector  $\mathbf{u}_t$ :

$$u_{t,i} = \begin{cases} 1 & \text{if } h_t(x_i) = y_i \\ -1 & \text{otherwise.} \end{cases}$$

If  $\mathbf{d}_t \cdot \mathbf{u}_t \leq 0$  then,  
set  $T = t - 1$  and abort this loop.  
Calculate distribution  $\mathbf{d}_{t+1}$ :  
Initialize  $\hat{\mathbf{d}}_1 = \mathbf{d}_1$ .  
For  $j = 1, 2, \dots$ :  
 $q_j = \operatorname{argmax}_{q_j \in \{1, 2, \dots, t\}} |\hat{\mathbf{d}}_j \cdot \mathbf{u}_{q_j}|$ .  
 $\hat{\alpha}_j = \ln \left( \frac{1 + \hat{\mathbf{d}}_j \cdot \mathbf{u}_{q_j}}{1 - \hat{\mathbf{d}}_j \cdot \mathbf{u}_{q_j}} \right)$ .  
For all  $i \in \{1, 2, \dots, m\}$ ,  
 $\hat{d}_{j+1,i} = \frac{1}{Z_j} \hat{d}_{j,i} \exp(-\hat{\alpha}_j u_{q_j,i})$ ,  
where  $Z_j = \sum_{i=1}^m \hat{d}_{j,i} \exp(-\hat{\alpha}_j u_{q_j,i})$ .  
**Output** the final hypothesis:  
 $h_{fin}(x) = \operatorname{argmax}_{y \in Y} \sum_{t: h_t(x)=y} \log \frac{1-\epsilon_t}{\epsilon_t}$ .

**Fig. 2.** Totally Corrective Boosting algorithm:  $\{(x_1, y_1), \dots, (x_m, y_m)\}$  is the training set,  $L_b$  is the base model learning algorithm, and  $T$  is the maximum allowed number of base models.

examples misclassified by the current base model have their weights increased for the purpose of creating the next base model, while correctly-classified training examples have their weights decreased. More specifically, if  $h_t$  misclassifies the  $i$ th training example, then its new weight  $d_{t+1,i}$  is set to be its old weight  $d_{t,i}$  multiplied by  $\frac{1}{2\epsilon_t}$ , where  $\epsilon_t$  is the sum of the weights of the examples that  $h_t$  misclassifies. AdaBoost assumes that  $L_b$  is a *weak learner*, i.e.,  $\epsilon_t < \frac{1}{2}$  with high probability. Under this assumption,  $\frac{1}{2\epsilon_t} > 1$ , so the  $i$ th example's weight increases ( $d_{t+1,i} > d_{t,i}$ ). On the other hand, if  $h_t$  correctly classifies the  $i$ th example, then  $d_{t+1,i}$  is set to  $d_{t,i}$  multiplied by  $\frac{1}{2(1-\epsilon_t)}$ , which is less than one by the weak learning assumption; therefore, example  $i$ 's weight is decreased. Note that  $\mathbf{d}_{t+1}$  is already normalized:

$$\begin{aligned}\sum_{i=1}^m d_{t+1,i} &= \frac{1}{2\epsilon_t} \sum_{i=1}^m d_{t,i} I(h_t(x_i) \neq y_i) + \frac{1}{2(1-\epsilon_t)} \sum_{i=1}^m d_{t,i} I(h_t(x_i) = y_i) \\ &= \frac{1}{2\epsilon_t} \epsilon_t + \frac{1}{2(1-\epsilon_t)} (1 - \epsilon_t) = 1.\end{aligned}$$

Under distribution  $\mathbf{d}_{t+1}$ , the total weight of the examples misclassified by  $h_t$  and those correctly classified by  $h_t$  become 0.5 each. This is done so that, by the weak learning assumption,  $h_{t+1}$  will classify at least some of the previously misclassified examples correctly. As shown in [1], this weight update scheme is equivalent to the usual scheme [5] but is intuitively more clear. The loop continues, creating the  $T$  base models in the ensemble. The final ensemble returns, for a new example, the one class in the set of classes  $Y$  that gets the highest weighted vote from the base models.

For all the base models  $h_t$  ( $t \in \{1, 2, \dots, T\}$ ) and the  $m$  training examples, construct a vector  $\mathbf{u}_t \in [-1, 1]^m$  such that the  $i$ th element  $u_{t,i} = 1$  if  $h_t$  classifies the  $i$ th training example correctly ( $h_t(x_i) = y_i$ ) and  $u_{t,i} = -1$  otherwise. Kivinen and Warmuth [7] pointed out that AdaBoost calculates  $\mathbf{d}_{t+1}$  from  $\mathbf{d}_t$  such that  $\mathbf{d}_{t+1} \cdot \mathbf{u}_t = 0$ . That is, the new distribution is created to be orthogonal to the mistake vector of  $h_t$ , which can be intuitively described as wanting the new base model's mistakes to be uncorrelated with those of the previous model. This naturally leads to the question of whether one can improve upon AdaBoost by constructing  $\mathbf{d}_{t+1}$  to be orthogonal to the mistake vectors of *all* the previous base models  $h_1, h_2, \dots, h_t$  (i.e.,  $\mathbf{d}_{t+1} \cdot \mathbf{u}_q = 0$  for all  $q \in \{1, 2, \dots, t\}$ ). However, there is no guarantee that a *probability distribution*  $\mathbf{d}_{t+1}$  exists that satisfies all the constraints. Even if a solution exists, finding it appears to be a very difficult optimization problem [7]. The Totally Corrective Algorithm (figure 2) attempts to solve this problem using an iterative method. The initial parts of the algorithm are similar to AdaBoost. That is, the Totally Corrective Algorithm uses the same  $\mathbf{d}_1$  as AdaBoost in creating the first base model and the next statement checks that the base model error is less than 0.5. The difference is in the method of calculating the weight distribution for the next base model. The Totally Corrective Algorithm starts with some initial distribution such as  $\mathbf{d}_1$ . It then repeatedly finds the  $q_j \in \{1, 2, \dots, t\}$  yielding the highest  $|\mathbf{d}_j \cdot \mathbf{u}_{q_j}|$ , and then











